NATIONAL OSG NEWSLETTER

Editor : Gordon Bentzen
8 Odin Street,
SUNNYBANK Qld. 4109.
(07) 345-5141

DECEMBER 1988

NATIONAL OS9 USER GROUP NEWSLETTER

EDITOR : Gordon Bentzen
HELPERS : Bob Devries and Don Berrie

SUPPORT : Brisbane OS9 Level 2 Users Group.

The National OS9 group continues to grow. As each month passes we are gaining new members at the rate of two, three or four each month. It does seem that this operating system has a growing number of followers, it certainly deserves to have. OS9 offers to us many very powerful features, such as sharable devices, files and modules, plus multitasking etc. but then you have heard all this before haven't you, so let's get back to the National OS9 user group for a moment.

With new members joining each month we are faced with the question of subscription expiry dates, what should they be ?? As many of you would be aware, we elected to provide a JULY '88 newsletter from our own funds to give you a sample of the new National OS9 Newsletter, this was extended to also provide a copy for AUGUST '88. The result is that the majority of your subscriptions were then treated as commencement 1st. September, and this made things pretty easy to keep track of. The fact that some newer members have asked about backcopies has prompted some further thought on this delicate problem. To cut a long story short, we have elected to provide back copies to all current members so that we can still work on a common expiry date of 31st.AUG '89. This means that all current members will be entitled to a copy of the monthly newsletter commencing from SEPT '88.
A copy of the July and August '88 editions will also be provided so that all will have the coplete set as it were. This will mean some reprinting which is currenty being organised. We do expect some delays here, so you should receive this December edition prior to any back copies due, but don't worry all should be mailed well before the end of January.

With the year of 1988 very rapidly drawing to a close, many of us will have thoughts of Christmas celebrations, a holiday away from the work place or the normal routine but we will also be preparing plans to carry us into a brand new year. As you would expect, some of this will affect our Newsletter, and the first is that we are planning material for our FEB 1989 edition. Yes you'e right, there will be not be a JANUARY 1989 edition.

Now the Good News! We are able to bring you a new article on 'C' thanks to Ross McKay who has sent in a very interesting article "Making the Most of Microware C - Part I" The receipt of this material from --Rosko!-- caused the total disruption of one of our newsletter planning meetings. It was a case of MUST READ before we did anything else. We are sure that this will interest many of our readers as well, and we can't wait to see a Part II. This is great material Rosko, many thanks for sharing it with us.

MERRY CHRISTMAS and a HAPPY, SAFE & PROSPEROUS NEW YEAR to ALL

Making the Most of Microware C - Part I

by --Rosko!--

NOTE: requires some knowledge of C and assembler (6809 or 68000)

Microware's C compiler for OS-9/6809 was made at a budget price for budget systems, yet designed to take advantage of the powerful 6809 (thanks to James McCosh). This meant that a good compiler was made available at an affordable price. What it also meant was that Microware couldn't afford to take too much time on the C library, so they wrote all of the system-independant stuff in C, and when they ported the C compiler to OS-9/68000 they left this in C.

Now don't get me wrong, I have nothing against C generally, otherwise I wouldn't use it so much. C compilers still produce machine code programs, but they can't match code written in tight assembly language for raw speed (not to mention code size). Many of the functions in the C library are used often in C programs, and their efficiency can greatly affect the efficiency of the programs using them. So I set out to discover the secrets of writing C library functions in assembly language... and discovered that it really is quite easy!

Before you rush off to merrily rewrite all of your C functions in assembler, there are a couple of points you should heed:

1. 6809 C reserves registers U, Y and S so you must restore them to their initial values on return (RTS) if you modify them. Likewise for 68000 C with D4-D7,A2-A7.

2. If you are going to rewrite existing C functions, you must rewrite any additional functions associated with them in the C library... the library is grouped into modules, you can examine them using 'roump'. (For example all the standard string functions, including strlen(), strcpy(), strcat() and strcmp() are in one module). If you just want to try something out, use a different name.

OK, so now we have the Tabu's out of the way, how does our assembler function receive arguments? If we write a function to find the length of a string, how do we get the pointer to the string? Well, in 6809 C all arguments are passed on the stack, so that when a function is BSR'ed to, it executes with S pointing to the return address, S+2 to the first argument, and so on. It does this by pushing arguments last-to-first onto the stack before calling the function.

eg1. func(arg1) produces a stack like:

```
        |     :     |
        | arg1 LSB  |   3,S
        | arg1 MSB  |   2,S
        | RTS addr LSB|  1,S
S -> | RTS addr MSB|   0,S
```

Note that C handles everything at word-size, i.e. 2 bytes for the 6809, except with long integers (4 bytes) and doubles (8 bytes).

In 68000 C, because there are 8000 many registers, the first 8 bytes of the arguments are passed in registers D0/D1, and the rest on the stack in the same fashion as 6809 C (but word size is 4 bytes). For the above example, the argument would be passed in D0. The exceptions are doubles, which are passed on the stack.

To access arguments on the stack, you offset the stack pointer by the word size (2 bytes 6809, 4 bytes 68000) to allow for the RTS address, plus the number of bytes for preceding arguments. For the above example, on the 6809 you would LDD 2,S to load the word 2 bytes above the bottom of the stack. If there were 4 integer arguments, to access the 4th argument you would LDD 8,S (6809) or MOVE.L 16(SP),D0. Note that if you change the stack pointer at all, for example by pushing registers (PSHS or MOVE.L ...,-(SP)) or by allocating stack space for variables, you must change the offset to arguments accordingly. For example, on the 6809 if you PSHS U and want to access the first argument, it is now at 4,S.

That now lets us pass an argument, but what about returning an answer? C allows only one return value per function, and it must be in one of the basic data types (int, long, pointer, double). This is so that return values can be put into the default data register. On the 68000, all return values are returned in D0 except doubles which are returned in D0/D1. On the 6809, int and pointer arguments are returned in D, but doubles and longs are funny; they are returned on the stack, with a pointer to them returned in X. Note that C converts arguments of type char and short to int, and float to double. READ LAST SENTENCE AGAIN.

OK, now we know enough to write a C function. Something easy is in order to begin with, so lets write a function to get the length of a string. length() is passed a pointer to a string, terminated with a NULL (0), and returns as an integer the length of the string. It's really a clone of strlen.

```
* 6809 version
* int length (s)      char *s;
        psect length_a,0,0,0,0,0
length: ldx   2,s     get pointer in X
length1 tst   ,x+     look for NULL
        bne   length1
        leax  -1,x    don't count the NULL
        tfr   x,d     get end address in D
        subd  2,s     subtract start address
        rts
        endsect

* 68000 version
* int length (s)         char *s;
        psect   length_a,0,0,0,0,0
length: movea.l d0,a0    get pointer in A0
length1 tst.b   (a0)+    look for NULL
        bne.s   length1
        subq.l  #1,a0    don't count NULL
        sub.l   a0,d0    subtract end from start address
        neg.l   d0       get sign right
        rts
        endsect
```

Type in the version for your machine, and save it in a file called length.a. To use it from a C program, you just call it with

```
x = length (strpointer);
```

as you would to call strlen(), and compile your C program with the following syntax, assuming fred.c

```
cc fred.c length.a -f=fred
```

(OS-9/6809 users use cc1 or cc2 if you haven't patched the compiler yet :-)

Well, I'm thoroughly sick of typing now, so bye till next time.

--Rosko!--

STARTING OUT WITH OS9

Modifying your system disk.

For most people, this seems to be a daunting task at the best of times. How do you do it ? Well, the information is right there in the manual. Of course, I'll admit to there being other and perhaps easier ways to do the same thing. Most people now-a-days would at least like to set up their OS9 for different disk drives. Let me say at the start of all this that I will be referring to two-disk systems only. To use OS9 on anything less is nothing less than excruciatingly painful, especially in the modification stage. If you don't have two disk drives, then try to get together with some-one who has, and please try to get another one yourself.

One of my main problems now is to keep referring to both the Level One and Level Two OS9 systems. I'll start out by talking about the Level One system briefly. Most modifications for Level One are at the grass-roots level, in that we would need to modify the device driver 'CCDISK' and the device descripter 'D0' and 'D1' using the programme 'DEBUG'. This is a fairly lengthy process and outside the scope of this part of my series. I will address this problem in the near future however. Suffice it to say, that if you have been subscribing to 'The Rainbow' from the USA then a patch file for making 'CCDISK' into double sided appeared in the October '88 issue of that magazine under the name 'DiskFix' and 'FormatFix'. These files are complete, and save the modified modules to your current data directory. If you then replace the original file 'CCDISK.DR' in the modules directory of the Level One disk, you can run the 'Config' programme to create a new disk. Don't forget to modify the device descriptors as well. More on that later.

I will now refer exclusively to Level Two systems. The second of the two disks contains a directory called 'MODULES'.In it there are all the necessary drivers and descripters to create a new system disk. The easiest (and slowest) way to create the new system is to format a disk for 35 track single sided and use it to produce a new boot disk with the new disk drivers on it. Use the 'Config' programme to install these. For those of you with 128K CoCo 3 machines, you will probably get an error when the config tries to call the 'OS9Gen' programme. DONT PANIC ! This is not an irretrieveable situation! When you get this situation, type this line:-

```
OS9GEN /D1 </D0/MODULES/BOOTLIST
```

This will rerun the same command that Config was trying to do when it ran out of memory space. When running by itself, it has enough memory to complete its task.

Now you have a 35 track disk that has 40 or 80 track descriptors on it. Don't worry! They are compatible, as a higher track size descripter can read the lower track size disks. This is all done in the 'CC3DISK' device driver. When you boot up OS9 with this disk, you will be able to format a disk with your specified number of tracks and sides. When you have done that, it is an easy task to use the command 'COBBLER' to put your current boot on this new disk. Just type 'COBBLER /D1' and there you are, a disk with the OS9Boot file on it. Now you can copy all your files from your boot disk onto it. An easy way to do this is to type this command :-

```
DSAVE /D0 /D1 ! SHELL
```

After some time all your files will be transferred from your source to your destination disk. Oh, by the way, it'll look like the computer is giving itself commands, which is exactly what dsave does do.

Of course, there are going to be some questions about all this. I am going to anticipate at least one of these, and that is 'where is the 80 track descriptor for drive /D0 ?'. Well you're right there isn't one, we'll have to make one. Unfortunately, this will require the 'save' command (not on the Level Two disks) and the modpatch command (which is). For those of you who have access to the Level One 'save' utility, you may safely use it. Those that don't, it is available on the OS9 development disk. You could also probably arrange something through this or your local user group. (Ring me or drop me a line if you get into trouble)

Now first of all, these commands must be in memory beforehand !!!

SAVE	for saving the new module, and
MODPATCH	to modify the old one.

So firstly type LOAD SAVE MODPATCH

You'll need to save the new module onto the disk in drive /D1 because you'll be changing the descriptor for drive /d0 and I don't think OS9 would like that. Now we get to the patching part. Type these commands exactly as they are here:-

```
modpatch
I d0
c 14 00 03
c 16 01 03
c 17 23 50
c 19 01 02
v
```

Now press CTRL and BREAK together. You'll notice that modpatch echoes some of your characters back at you, this is normal. Now it is time to save the module. type this:-

```
SAVE /d1/d0_80d.dd d0
```

When this is finished, reboot the system, copy the new module into the modules directory of the D2 disk and go through the first part of this section to create the 80 track bootable disk. By the way, this disk will be bootable from disk basic. Make sure your drives are set for 80 track working if they're switchable.

So now you have a system with the disk drivers to your satisfaction. If there are any questions, please feel free to ring me or drop me a line, and I will try to answer your questions.

Bob Devries.

Reading Logical Sector Zero (LSN0)

Whilst in conversation with Don the other day, I felt the need for a programme to print out the contents of the first part of the first sector of the disk (LSN0) along with a short explanation of what the numbers all mean. So, I spent a few hours writing a programme in C (can't you see its my favourite ?) to do just that. And after a moderately long debugging session (the compiler and I didn't agree on how to print hexadecimal) I came up with this programme which I called LSN0.

The command line format is LSN0 /devicename

If you get it wrong, it'll jog your memory to do it right.

The source for the programme is on the following pages. Mostly it just opens the disk as a file, reads a number of characters from it (that number is set by the length of the structure ddsect) and then closes the file again. It then proceeds to print the titles for the data it retrieved, and the data in decimal or hexadecimal or string format, whichever suits the need.

The programme tells you what is printed in the Level Two manual on page 5-2 of the 'OS9 Technical Reference' section. It has the explanations in English instead of computer, and may be of use to some-one.

Bob Devries

```
#include (stdio.h)
#include (direct.h)
union sec {
        long numsec;
        char tsec[4];
        unsigned usec;
        };

main (argc,argv)
int argc;
char *argv[];
{
        int fp;
        struct ddsect disk;
        int i,count;
        union sec totsec;

        pflinit();

        for (i=0;i(4;i++) totsec.tsec[i] = '\0';
        if (argc != 2)
        {
        useage();
        exit(0);
        }

        if (argv[1][0] != '/')
        {
        useage();
        exit(0);
        }

        count = strlen(argv[1]);

        for (i=1;i (= count; i++)
        {
        if (argv[1][i] == '/')
                {
                fprintf(stderr,"%s is not a device name.\n\n",argv[1]);
                useage();
                exit(0);
                }
        }
argv[1][count++] = '@';
argv[1][count] = '\0';

if ((fp = open(argv[1],_READ)) == EOF)
        {
        fprintf(stderr,"Can't open %s for reading.\n\n",argv[1]);
        useage();
        }
read(fp,&disk,sizeof(disk));
close(fp);

totsec.tsec[1] = disk.dd_tot[0];
totsec.tsec[2] = disk.dd_tot[1];
```

```
totsec.tsec[3] = disk.dd_tot[2];

printf("\n LSN 0 description of %s.\n",argv[1]);
printf("\n\n Total number of sectors = %ld.\n",totsec.numsec);
printf(" Sectors per track = %d.\n",disk.dd_tsk);
printf(" Bytes in allocation map = %u.\n",disk.dd_map);
printf(" Cluster size = %u.\n",disk.dd_bit);

totsec.tsec[1] = disk.dd_dir[0];
totsec.tsec[2] = disk.dd_dir[1];
totsec.tsec[3] = disk.dd_dir[2];

printf(" Address of Root Directory = %ld.\n",totsec.numsec);
printf(" Owner number = %u.\n",disk.dd_own);


totsec.tsec[0] = '\0';
totsec.tsec[1] = disk.dd_att;

printf(" Attributes = %02X.\n",totsec.usec);
printf(" Disk ID = %u.\n",disk.dd_dsk);
printf(" Media format = %d ",disk.dd_fmt);
switch(disk.dd_fmt)
        {
        case 0:
                printf("48 tpi SDSS.\n");
                break;
        case 1:
                printf("48 tpi SDDS.\n");
                break;
        case 2:
                printf("48 tpi DDSS.\n");
                break;
        case 3:
                printf("48 tpi DDDS.\n");
                break;
        case 4:
                printf("96 tpi SDSS.\n");
                break;
        case 5:
                printf("96 tpi SDDS.\n");
                break;
        case 6:
                printf("96 tpi DDSS.\n");
                break;
        case 7:
                printf("96 tpi DDDS.\n");
                break;
        default:
                printf("     Can't identify format.\n");
        }

printf(" Sectors per track (track 0) = %u.\n",disk.dd_spt);

totsec.tsec[1] = disk.dd_bt[0];
totsec.tsec[2] = disk.dd_bt[1];
totsec.tsec[3] = disk.dd_bt[2];
```

```
        printf(" Address of bootstrap file = %ld.\n",totsec.numsec);
        printf(" Length of bootstrap file = %u",disk.dd_bsz);
        if (disk.dd_bsz == 0)
                printf(" No boot installed.\n");
        else
                printf(".\n");

        printf(" Creation date : %02d/%02d/%02d %02d:%02d.\n", disk.dd_date[0], disk.dd_date[1], disk.dd_date[2],
disk.dd_date[3], disk.dd_date[4]);
        for (i=0; i<32; ++i)
                {
                if (disk.dd_name[i] < 0)
                        {
                        disk.dd_name[i] -= 128;
                        disk.dd_name[++i] = '\0';
                        break;
                        }
                }
        printf(" Volume name = %s.\n\n",disk.dd_name);

}

useage()
{
        fprintf(stderr,"\n Useage: lsn0 /devicename\n\n");
}
```

RUNNING A TERMINAL UNDER OS9 : A TUTORIAL FOR THE COCO

I guess, like almost everybody else, you would at least like to see if it is possible to run a terminal on your CoCo. You will no doubt have noticed a number of commands in the development pack which are used for communication. Specifically they are Tsmon, and Login. But does it work?

Well yes it does, and quite well. There are however, some limitations, particularly when transferring files between machines. However that is an issue for later on. First, let's look at some different setups.

Before we go any further, it really is a necessity to have some type of genuine RS232 interface, either the so called Deluxe RS232 interface pack or equivalent. I believe that one of our members has developed his own version of that pack which he mounts internally. Hopefully we might be able to persuade him to make some of the hardware available for sale, or perhaps to provide details so that members can make their own. You will NOT be able to use the CoCo's built in serial port at the back of the machine. This means that in most cases, you will need a Multi-Pak Interface as well! Still with us?

If you do decide to try to use the built in serial port, you will find that it occupies so much of the system resources, that there will be almost no processor time available to service any other processes. I really don't know why Microware included the driver and descriptor for /T1 in the system.

Other requirements will be, a terminal (obviously), a null-modem cable, or a cable to your modem, and some type of terminal programme for the remote terminal. The wiring of null-modem cables and the like has been described in a number of other publications, particularly the US Rainbow, so we won't deal with it here.

Now to get the terminal running, we need to have the proper driver and descriptor in memory. (I use /T2 and ACIAPAK, but there is no reason why you can't use the /T3 modules.) If you don't already have them in memory, it makes sense to merge them into one file before you load them so that they only occupy one 8K block. Use the commands :

```
Merge /D0/MODULES/t2.dd /D0/MODULES/aciapak.dr >/D0/t2pak
Attr t2pak e pe
Load /D0/t2pak
```

The next thing we need to do is to ensure the baud rates of both the terminal and the terminal port set the same. You will have to set the baud rate for the terminal using whatever commands are required in the terminal program. In some cases, it may be hard coded into the programme (eg XCOM9). Set the /T2 baud rate by using the command :

```
Xmode /T2 baud=x    (where x is from 1 to 7)
                    (1 equals 300 baud     )
                    (7 equals 19.2 Kbaud   )
```

Once the baud rates are the matched, we are ready to start.

In order for the terminal to be able to access the system, we need to give it some resources. We do this by starting a shell on the communications port. There are two ways to achieve this. We can use the Tsmon command to do it automatically or we can manually start a shell by redirecting a shell to /T2.

Tsmon is started by the command :

```
Tsmon /T2&
```

Once the process is started, it runs as a background task until it detects an incoming <CR>. When this happens, it wakes up, and immediately runs Login. Login then prompts the remote terminal for a User Name, and then for a Password. It compares the answers given with those listed in the file /D0/SYS/password, and will look for that file on the drive upon which the system was first booted. In other words, for Tsmon to function successfully, you will need to leave your boot disk in its drive.

The obvious use for Tsmon is to allow other users to log onto your system by the use of an external modem. In effect, a very simple Bulletin Board! A word of caution however. You will need to edit the /D0/SYS/password file so that only those people that you want to be able to access your system can do so. It is particularly important that you do not allow anyone to log on as User 0, as there are no limits to the actions of that user, even if logged on to the system via /T2. (Including reformatting disks, looking at personal mail files etc !!!). Be ultra careful if you have an auto-answer modem! Also make sure that the more "dangerous" commands do not have the public execute (pe) attributes set.

The other way of starting a shell on /T2 is by the use of redirection. Try the following :

```
Shell <>>>/T2&  or perhaps better : Shell i=/T2&
```

The first of these two commands will die when an EOF is sent, and will have to be restarted again from the keyboard of the host. The second will automatically restart on EOF. This method of starting the terminal shell would be most suitable when you have another user connected via a null-modem cable.

I regularly use my XT clone as a terminal to my CoCo running at 19.2 Kbaud, and it works really well. In fact, when running at that speed, the screen updates are much faster than those of the CoCo. I also find the ability to log onto another users system a great advantage, and Bob Devries and I regularly use each others systems via our modems. (It's a bit slow at 300 baud, but it does work, and we find it most convenient.)

Next month, I will go further into the use of terminals, particularly with reference to file transfer using the OS9 version of Kermit. Until then, if you need any assistance, call me on (07) 375 3236, or Bob Devries on (07) 372 7816 and we will try to be of assistance.

CORRECTIONS TO ZAP

After trying to run ZAP on a system with a 10 mB hard disk, a number of problems arose. While the original programme does work, it is not particularly elegant in its treatment of the reading of LSN 0. The following changes will fix the problem of reading a disk which has more than 32768 sectors. (A 10 mB hard disk has more than 39000 sectors!). Changes have also been incorporated to make the programme more elegant.

You will need to modify three modules, Zap, Getsec, and Getdev. The changes are all commented in the following listings, and the changes should appear **BOLD** in the final printing.

```
PROCEDURE zap
  BASE 0
  DIM PAGE,PATH,wpath,fgnd,bgnd,bord:BYTE
  DIM maxblock:REAL (* maxblock needs to be a real value to allow for *)
  DIM ident,x,y,x1,y1:INTEGER            (* hard disk sector numbers *)
  DIM title:STRING[40]
  DIM CHOICE:STRING[25]
  .
  .

  PRINT #wpath,"       Version 1.11 88/11/25" (* Change version number and date *)
  PRINT #wpath,
  PRINT #wpath,"## USE WITH CARE - PERMANENT CHANGES TO DISK STRUCTURE CAN BE MADE ##"
  PRINT #wpath,
  RUN getdev(wpath,NAME,PATH,secdat,maxblock,ident)
  PRINT #wpath,
  RUN getsec(wpath,maxblock,BLKNO,secdat) (* Pass sector data to getsec *)
  ON ERROR GOTO 1000
10 METER=BLKNO/256
  iblkno=BLKNO
  COUNT=0
  SEEK #PATH,30 (* Start at byte 0 of LSN 0 *)
  GET #PATH,secdat
  IF secdat(14)*256+secdat(15)<>ident THEN
  x=6 \x1=40 \y=8 \y1=8
  .
  .

  IF CHOICE="D" OR CHOICE="d" THEN
  x=5 \x1=30 \y=10 \y1=6
  RUN winopen(wpath,x,y,x1,y1)
  RUN getdev(wpath,NAME,PATH,secdat,maxblock,ident)
  RUN getsec(wpath,maxblock,BLKNO,secdat) (* Pass sector array *)
  GOTO 10
  ENDIF
  .
  .

PROCEDURE getsec
```

```
  BASE 0   (********** This line is an addition *)
  PARAM wpath:BYTE
  PARAM maxblock:REAL (*** maxblock redefined as a real value *)
  PARAM blkno:REAL
  PARAM secdat(256):BYTE (*** Pass values for current sector *)
  DIM hblkno:STRING[25]
1 PRINT #wpath,"SECTOR NUMBER (max:$";
  PRINT #wpath USING "H2>",secdat(0);   (* These 3 lines replace   *)
  PRINT #wpath USING "H2>",secdat(1);   (* the line beginning with *)
  PRINT #wpath USING "H2>",secdat(2)-1; (* PRINT #wpath USING ...  *)
  PRINT #wpath,") :";
  INPUT #wpath," ",hblkno
  hblkno="$"+hblkno
  ON ERROR GOTO 1
  blkno=VAL(hblkno)
  IF blkno>maxblock OR blkno<0 THEN
     GOTO 1
  ENDIF
  RUN winclose(wpath)
  END

PROCEDURE getdev
  BASE 0
  ON ERROR GOTO 10
  PARAM wpath:BYTE
  PARAM name:STRING[4]
  PARAM path,secdat(256):BYTE
  PARAM maxblock:REAL (* maxblock redefined as a real value *)
  PARAM ident:INTEGER
5 INPUT #wpath,"RBF Device Name : ",name
  IF LEFT$(name,1)<>"/" THEN
     name="/"+name
  ENDIF
  IF RIGHT$(name,1)<>"@" THEN
     name=name+"@"
  ENDIF
  OPEN #path,name
  SEEK #path,0 (*** Change seek byte to the start of LSN0 *)
  GET #path,secdat
  maxblock=secdat(0)*65536.+secdat(1)*256+secdat(2)-1 (* Determine value   *)
  ident=secdat(14)*256+secdat(15)                     (* using all 3 bytes *)
  END                                                 (* of DD.TOT         *)
10 PRINT #wpath,"*** - DEVICE NAME REQUIRED"
  GOTO 5
```